Spring 2015

Creating a Mobile Game

Timothy Jasany
The University Of Akron, trj21@zips.uakron.edu

Creating an Android Game: Lunar Strike

Tim Jasany

Department of Computer Science

Honors Research Project

Submitted to

The Honors College

Approved:

Kathy J. Liszka Date 4/20/15
Honors Project Sponsor (signed)

Kathy J. Liszka
Honors Project Sponsor (printed)

Date 4/20/15
Reader (signed)

Zhong-Hui Duan
Reader (printed)

Date 4/21/2015
Reader (signed)

Tim O'Neil
Reader (printed)

Accepted:

TSN Date 4/22/15
Department Head (signed)

T.S. NORFOLK
Department Head (printed)

Date 4/21/2015
Honors Faculty Advisor (signed)

Tim O'Neil
Honors Faculty Advisor (printed)

Date
Dean, Honors College

Tim Jasany

Honors Research Project

Creating a Mobile Game

Application Title: Lunar Strike

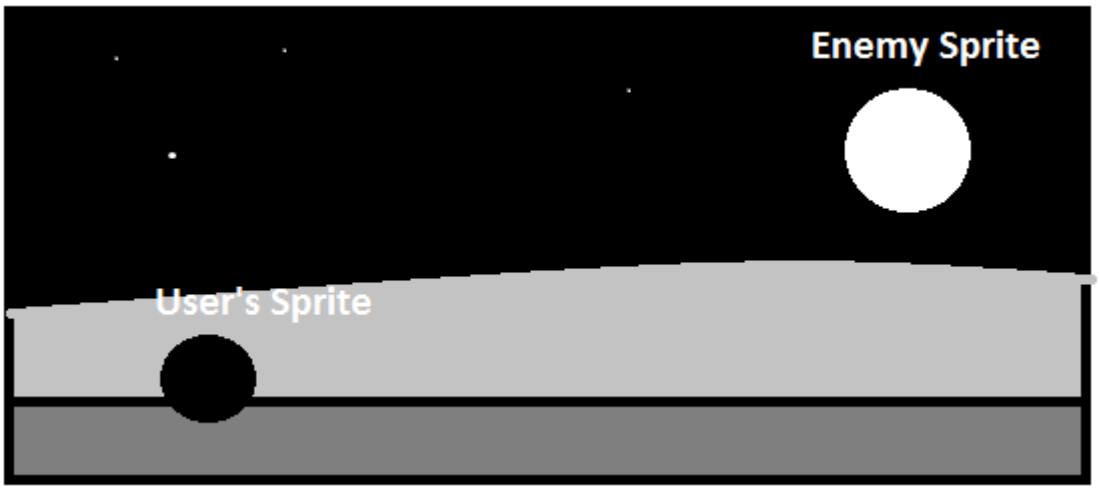
In today's world, the use of mobile devices is on the rise. Whereas not every person has a laptop or a computer at his or her house, most people have a mobile device. This is due to the fact that mobile devices are becoming a common day necessity, being the main way many people communicate with one another. Whether it is someone communicating through an email for a business or someone tweeting on twitter, most people today use a mobile device for communicating. As time has gone by, mobile devices have become more and more advanced. No longer are mobile devices just a way to call someone or send an email, now mobile devices can be used for basically anything ranging from managing your bank account to playing a simple game to pass the time. Since mobile devices have become so important in today's everyday life, I decided it would be in my best interest to learn mobile programming, and decided to design my first real app. Having game programming as my interest in my field of study, I determined I would program a mobile game.

I decided to program my game using the IDE known as Android Studio. I also wrote my program in Java and XML. The Java is used for the background workings of the game, and the XML is used for the visuals of the game. The app is also made mostly using Android libraries.

I decided to create a free running game while considering the different genres of games I could design. I wanted to create a free running game that was not like most of the other free running games on the Google Play App Store. My design was to create a game where you are continuously trying to dodge an enemy that kept trying to collide with you as you ran through the game. I also wanted to

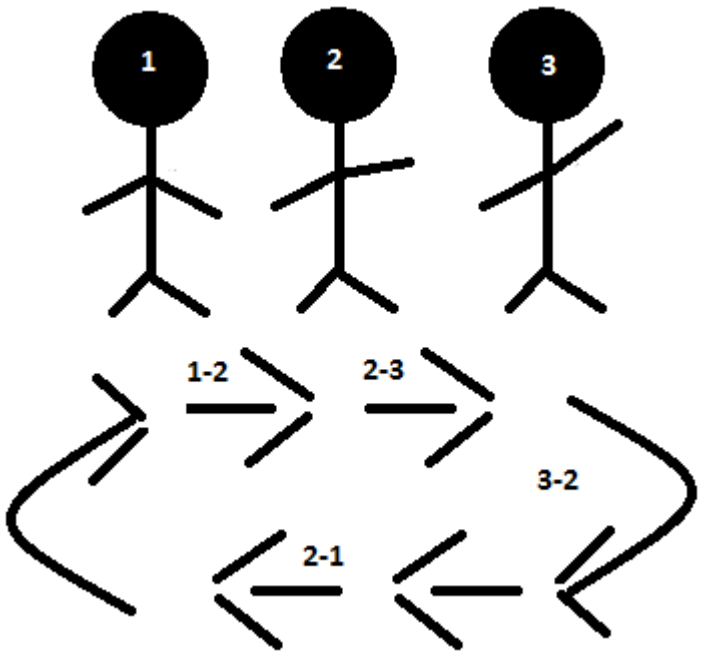
make the game so that it was actually hard, so I decided to make the enemy have different phases and also have a random aspect to it, so that it is unpredictable. The basic design of the player is that the player could move right and left and also jump; however, the player could not move outside the boundaries of the screen. If at any point the player collides with the enemy, the game ends, and you can either choose to play again or exit the game. The way I designed the enemy to be unpredictable was that it would either be in one of two phases. The two phases for the enemy are to either be going from the left to the right then right to left once the enemy went off the screen, or be going from the top to the bottom then bottom to top. Since I wanted the enemy to be unpredictable, I designed it so he would appear at a random width or height each time he went off the screen. This means that there is no way to know where the enemy will be next time he comes on screen. I also made the enemy gain speed for each certain distance milestone reached. This means as the player gets further, the enemy will get faster and more difficult to dodge. To make the game feel more like a free runner, I wanted to have a moving background so that it would appear as if the player is really moving. Additionally, I wanted to make the game feel even more like you were moving, so I used a technique known as a parallax scrolling background. I achieved this effect by creating three separate backgrounds: one for the ground, one for the background, and one for the sky, which is space in my game. To give the illusion of moving with the three backgrounds, I moved the ground background at a pace similar to the speed of the player. I then made the background move at a slower speed than the ground, and also made the sky move at an even slower speed than the background. Having the three backgrounds move at different speeds allows for the player to think he or she is actually moving at real relations to the backgrounds behind him.

Sketch of original design



I also designed the game so that the player's sprite would be animated. To make the player have an animated sprite, I used a sprite sheet that was drawn by my friend. The sprite sheet consists of seven images that the player cycles through. There are six running animation images and one image of the player jumping. The game is able to cycle through the images because I made a Boolean that lets the sprite know if it is off the ground, so it knows if it should be using the jump image or cycling through the run images.

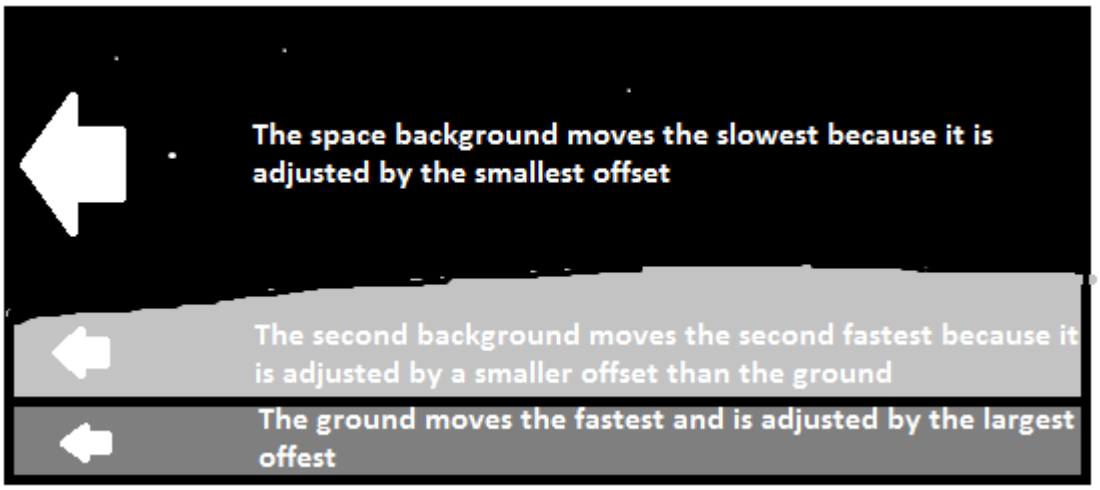
Diagram how sprite animation works



The Program works from a hierarchal stand point that the application is considered an activity in the Android framework. The activity basically is the app as a whole and is the starting point of execution. When the activity is created, the OnCreate internal method is called which I used to create a new view. The view then creates its own separate worker thread that is used to update the internal working of the game and also to update the screen and what should be displayed. The game uses a surface view in Android which I put a canvas on. Using this technique, I was able to create a touchable screen that I can draw to, and that the user can interact with. The game is constantly being updated and keeping track of specific events. The first part of the game that is constantly updated and drawn is the background of the game. The background is actually three separate backgrounds that I drew that overlap one another. The backgrounds move at different speeds to create an illusion of movement through a perspective from different distances. To draw the images onto the screen, I would update

their position then draw each background in the correct order. I made the backgrounds move by creating offsets that would be used to move the position of backgrounds horizontally by a constant number corresponding to each specific background. I set the offset of the closest background as the largest number so that the ground would appear to be moving the fastest, then set the offset of the second background slower than the first so it would appear to be moving slower. The third background that is furthest from sprite in the game then has the smallest offset so that it appear to be moving much slower than the ground.

Conceptual view of the moving backgrounds



I created a class for the sprite that contained its position, speed, direction, and also the image for the sprite. The class also contained information such as the frame number for the animation of the sprite and also other information to animate the sprite at a constant frames per second (fps). The sprite's class also contains the functions to handle touch events if the sprite is touched, and the sprite's own update method. The sprite class also contains another class I created that handle's the sprite's speed. The movement class holds the speed the sprite is moving horizontally and vertically and also what direction the sprite is moving.

The design of the game is such that the user controls the sprite through the three buttons on the screen. When the left or right button is touched, the sprite's horizontal direction changes corresponding to the button touched. When the jump button is touched, the sprite jumps; however, the user must hold the jump button, which makes the sprite jump higher. The objective of the game is to see how far of a distance the user can make it while dodging the enemy sprite. The enemy sprite constantly moves across the screen and the user tries to avoid a collision with the enemy sprite. The game ends when the enemy sprite finally collides with the user's sprite, which then prompts the user to either play the game again or to close the application.

The pseudo code view point of the game is demonstrated below:

```
Application is started
    Activity is created
    View is created and worker thread is created
        The backgrounds are drawn and the sprites are created and drawn to the screen
    The game updates
        The backgrounds move corresponding to their offset and are redrawn
        The sprite's position is updated and the sprite is drawn
            Check is button is pressed
                Update sprite's direction and position
            Check if user sprite collided with enemy sprite
                Create alert of score and asks user to play the game again
```

Restart game or close application based on selection

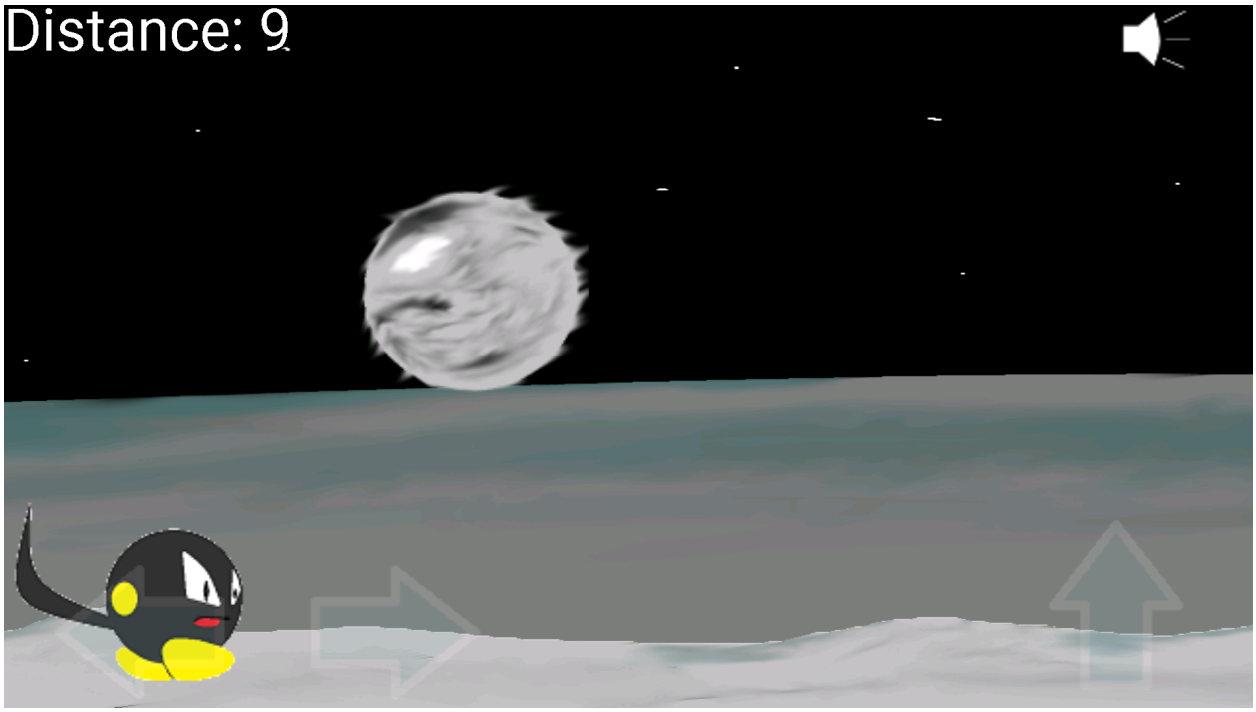
There were multiple issues that I ran into when creating the game. The one issue that took the most time to figure out was the problem of getting the background to loop correctly across the screen. I originally made the backgrounds reset position once the end of the image touched the width of the screen. Although this reset the background, it did not appear as though the ground was correctly looping. There would be a quick snap back of the image, and it was visual on the screen. I did a lot of research trying to find how to loop the backgrounds correctly, but when I tried different techniques such as doubling the image or creating two instances of the image, I was faced with memory errors and kept getting the error that I had run out of memory. I spent a lot of time thinking how to fix this issue, but I eventually realized that I could just draw the image to the screen twice when the end of the first image reached the width of the screen. I had not seen this answer anywhere, and I felt that it was strange no one else had thought of doing this technique. When I implemented the new version of the background updating method, I included drawing the image again at the end of the other image, and the issue did not appear anymore. The new implementation created a correctly looping background without running out of space and also made the game look a lot more aesthetically pleasing.

The other issue that took a great deal of time to work out had to deal with animating the sprite correctly. The first problem was that the entire sprite sheet would be drawn instead of a single image from the sprite sheet. This issue was due to the fact that to draw a single image from a sprite sheet, you use a method of two rectangles. The first rectangle is made by creating a rectangle “cut out” of the sprite sheet using the sprite’s height and width. The second rectangle is used to take the new “cut out” and place it at the correct position on the screen. Figuring out the rectangles was tricky and took some

work since it would just shrink my entire sprite sheet when I tried to draw a single image, but once I looked more into how the rectangles worked according to the sprite sheet, I was able to draw a single image instead of the entire sprite sheet. The other issue that the animated sprite had was that it would not follow the correct edge detection and collision detection as a regular sprite of a single imaged sprite. This issue was due to the fact that the detections were checking the original images width and height, which was causing many issues. Since the original image was seven times the width of a single sprite image, I changed the implementation so that it correctly checked the edge detection and collision detection based on the width of a single sprite image, instead of the entire width of the original sprite sheet. This solution lead to the resolution of the problem.

The program works as I intended it to, and is a fully playable game. The backgrounds loop correctly, and the sprite moves correctly based on the user's input. I also made sure that the app closes correctly and kills its own process when the app is closed. Furthermore, I implemented music into the game that I made myself using a synthesizer. The music can be toggled using the button I created located near the top of the screen the looks like a speaker. The game also correctly calculates and updates the user's distance traveled on the screen, and resets the distance data when the game is reset. The enemy sprite correctly randomly chooses a phase that indicates what direction it moves, and the enemy sprite also correctly moves across the screen at a random position indicated by the phase it is currently in. The collision detection between the user's sprite and the enemy sprite works as it was intended to. Overall the game runs properly and is playable, but the only complaint I have received is that the game is too challenging, which I had originally planned.

Screenshot of the game working



The future work I have planned for this game includes making the game more functional. The game right now only handles one touch event at a time, but not multi touch. I would like to implement multi touch that allows the user to touch multiple buttons at once, so the user is not restricted to only pressing a single button at a time. I believe the movement of the sprite would be less restricted using multi touch and would allow for a more interesting way to play the game. This change may also affect the difficulty as perceived by players.

References

"Android Studio Overview." *Android Studio Overview*. Web. 27 Feb. 2015. <http://developer.android.com/tools/studio/index.html>.

"Animoog | Moog Music Inc." *Animoog | Moog Music Inc*. Web. 7 Mar. 2015. <http://www.moogmusic.com/products/apps/animoog-0>.

"Android Tutorials." *Java Code Geeks*. Web. 7 Mar. 2015. <http://www.javacodegeeks.com/tutorials/android-tutorials/>.

"Parallax Scrolling: A Simple, Effective Way to Add Depth to a 2D Game - Tuts Game Development Tutorial." *Game Development Tuts*. Web. 9 Mar. 2015. <http://gamedevelopment.tutsplus.com/tutorials/parallax-scrolling-a-simple-effective-way-to-add-depth-to-a-2d-game--cms-21510>.

"Android Basics Archives - Mybringback." *Mybringback*. Web. 7 Mar. 2015. <http://www.mybringback.com/series/android-basics/>.